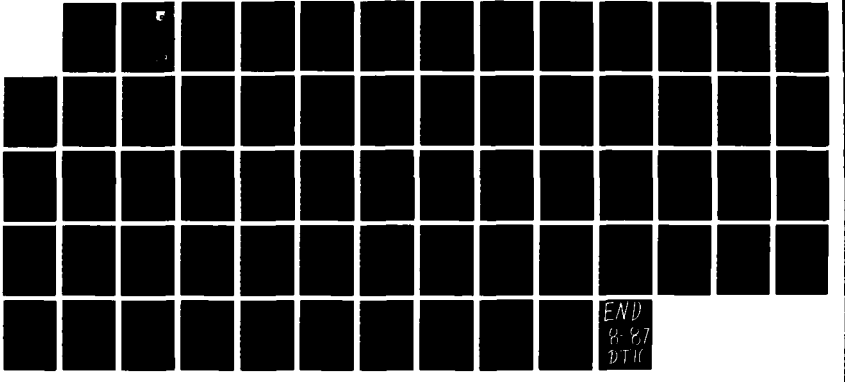
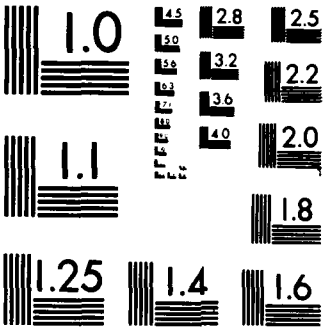
AD-A182 579 INTEGRATED INFORMATION SUPPORT SYSTEM (IISS) VOLUME 8 1/1
USER INTERFACE SUBS..(U) GENERAL ELECTRIC CO
SCHENECTADY NY PRODUCTION RESOURCES CONSU.. L JONES
UNCLASSIFIED 01 NOV 85 UM-6201443008 F/G 12/6 NL





AFWAL-TR-86-4006
Volume VIII
Part 11

AD-A182 579

INTEGRATED INFORMATION SUPPORT SYSTEM (IISS)
Volume VIII - User Interface Subsystem
Part 11 - Virtual Terminal User Manual

General Electric Company
Production Resources Consulting
One River Road
Schenectady, New York 12345

Final Report for Period 22 September 1980 - 31 July 1985

November 1985

Approved for public release; distribution is unlimited.

MATERIALS LABORATORY
AIR FORCE WRIGHT AERONAUTICAL LABORATORIES
AIR FORCE SYSTEMS COMMAND
WRIGHT-PATTERSON AFB, OH 45433-6533

NOTICE

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely related Government procurement operation, the United States Government thereby incurs no responsibility nor any obligation whatsoever; and the fact that the government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication or otherwise as in any manner licensing the holder or any other person or corporation, or conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

This report has been reviewed by the Office of Public Affairs (ASD/PA) and is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nations.

This technical report has been reviewed and is approved for publication.

DAVID L. JUDSON, PROJECT MANAGER
AFWAL/MLTC
WRIGHT PATTERSON AFB OH 45433

5 Aug 1986
DATE

FOR THE COMMANDER:

GERALD C. SHUMAKER, BRANCH CHIEF
AFWAL/MLTC
WRIGHT PATTERSON AFB OH 45433

7 Aug 86
DATE

"If your address has changed, if you wish to be removed from our mailing list, or if the addressee is no longer employed by your organization please notify AFWAL/MLTC, W-PAFB, OH 45433 to help us maintain a current mailing list."

Copies of this report should not be returned unless return is required by security considerations, contractual obligations, or notice on a specific document

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE

A182 579

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S) AFWAL-TR-86-4006 Vol VIII, Part 11
6a. NAME OF PERFORMING ORGANIZATION General Electric Company Production Resources Consulting 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION AFWAL/MLTC
6c. ADDRESS (City, State and ZIP Code) 1 River Road Schenectady, NY 12345	7b. ADDRESS (City, State and ZIP Code) WPAFB, OH 45433-6533
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Materials Laboratory Air Force Systems Command, USAF 8b. OFFICE SYMBOL (If applicable) AFWAL/MLTC	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F33615-80-C-5155
8c. ADDRESS (City, State and ZIP Code) Wright-Patterson AFB, Ohio 45433	10. SOURCE OF FUNDING NOS PROGRAM ELEMENT NO. 78011F; PROJECT NO. 7500; TASK NO. 62; WORK UNIT NO. 01
11. TITLE (Include Security Classification) (See Reverse)	
12. PERSONAL AUTHOR(S) Jones, Larry	
13a. TYPE OF REPORT Final Technical Report 13b. TIME COVERED 22 Sept 1980 - 31 July 1985	14. DATE OF REPORT (Yr., Mo., Day) 1985 November 15. PAGE COUNT 74
16. SUPPLEMENTARY NOTATION ICAM Project Priority 6201 The computer software contained herein are theoretical and/or references that in no way reflect Air Force-owned or -developed computer software.	

17. COSATI CODES		
FIELD	GROUP	SUB GR
1308	0905	

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This manual describes the program callable interface to the IISS Virtual Terminal, the Virtual Terminal commands, and provides terminal implementation information for programmers who wish to add new terminal types to the system.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT ☐ DTIC USERS ☐	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL David L. Judson	22b. TELEPHONE NUMBER (Include Area Code) 513-255-6976 22c. OFFICE SYMBOL AFWAL/MLTC

DD FORM 1473, 83 APR EDITION OF 1 JAN 73 IS OBSOLETE

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE

11. Title

Integrated Information Support System (IISS)
Vol VIII - User Interface Subsystem
Part 11 - Virtual Terminal User Manual

A S D 86 0029
9 Jan 1986

PREFACE

This user's manual covers the work performed under Air Force Contract F33615-80-C-5155 (ICAM Project 6201). This contract is sponsored by the Materials Laboratory, Air Force Systems Command, Wright-Patterson Air Force Base, Ohio. It was administered under the technical direction of Mr. Gerald C. Shumaker, ICAM Program Manager, Manufacturing Technology Division, through Project Manager, Mr. David Judson. The Prime Contractor was Production Resources Consulting of the General Electric Company, Schenectady, New York, under the direction of Mr. Alan Rubenstein. The General Electric Project Manager was Mr. Myron Hurlbut of Industrial Automation Systems Department, Albany, New York.

Certain work aimed at improving Test Bed Technology has been performed by other contracts with Project 6201 performing integrating functions. This work consisted of enhancements to Test Bed software and establishment and operation of Test Bed hardware and communications for developers and other users. Documentation relating to the Test Bed from all of these contractors and projects have been integrated under Project 6201 for publication and treatment as an integrated set of documents. The particular contributors to each document are noted on the Report Documentation Page (DD1473). A listing and description of the entire project documentation system and how they are related is contained in document FTR620100001, Project Overview.

The subcontractors and their contributing activities were as follows:

TASK 4.2

Subcontractors	Role
Boeing Military Aircraft Company (BMAC)	Reviewer.
D. Appleton Company (DACOM)	Responsible for IDEF support, state-of-the-art literature search.
General Dynamics/ Ft. Worth	Responsible for factory view function and information models.

Subcontractors	Role
Illinois Institute of Technology	Responsible for factory view function research (IITRI) and information models of small and medium-size business.
North American Rockwell	Reviewer.
Northrop Corporation	Responsible for factory view function and information models.
Pritsker and Associates	Responsible for IDEF2 support.
SofTech	Responsible for IDEF0 support.

TASKS 4.3 - 4.9 (TEST BED)

Subcontractors	Role
Boeing Military Aircraft Company (BMAC)	Responsible for consultation on applications of the technology and on IBM computer technology.
Computer Technology Associates (CTA)	Assisted in the areas of communications systems, system design and integration methodology, and design of the Network Transaction Manager.
Control Data Corporation (CDC)	Responsible for the Common Data Model (CDM) implementation and part of the CDM design (shared with DACOM).
D. Appleton Company (DACOM)	Responsible for the overall CDM Subsystem design integration and test plan, as well as part of the design of the CDM (shared with CDC). DACOM also developed the Integration Methodology and did the schema mappings for the Application Subsystems.

Subcontractors	Role
Digital Equipment Corporation (DEC)	Consulting and support of the performance testing and on DEC software and computer systems operation.
McDonnell Douglas Automation Company (McAuto)	Responsible for the support and enhancements to the Network Transaction Manager Subsystem during 1984/1985 period.
On-Line Software International (OSI)	Responsible for programming the Communications Subsystem on the IBM and for consulting on the IBM.
Rath and Strong Systems Products (RSSP) (In 1985 became McCormack & Dodge)	Responsible for assistance in the implementation and use of the MRP II package (PIOS) that they supplied.
SofTech, Inc.	Responsible for the design and implementation of the Network Transaction Manager (NTM) in 1981/1984 period.
Software Performance Engineering (SPE)	Responsible for directing the work on performance evaluation and analysis.
Structural Dynamics Research Corporation (SDRC)	Responsible for the User Interface and Virtual Terminal Interface Subsystems.

Other prime contractors under other projects who have contributed to Test Bed Technology, their contributing activities and responsible projects are as follows:

Contractors	ICAM Project	Contributing Activities
Boeing Military Aircraft Company (BMAC)	1701, 2201, 2202	Enhancements for IBM node use. Technology Transfer to Integrated Sheet Metal Center (ISMC).

Contractors	ICAM Project	Contributing Activities
Control Data Corporation (CDC)	1502, 1701	IISS enhancements to Common Data Model Processor (CDMP).
D. Appleton Company (DACOM)	1502	IISS enhancements to Integration Methodology.
General Electric	1502	Operation of the Test Bed and communications equipment.
Hughes Aircraft Company (HAC)	1701	Test Bed enhancements.
Structural Dynamics Research Corporation (SDRC)	1502, 1701, 1703	IISS enhancements to User Interface/Virtual Terminal Interface (UI/VTI).
Systran	1502	Test Bed enhancements. Operation of Test Bed.

TABLE OF CONTENTS

SECTION 1

INTRODUCTION

This manual describes the program callable interface to the Integrated Information Support System Virtual Terminal, the Virtual Terminal commands, and provides terminal implementation information for programmers who wish to add new terminal types to the system. Although the program callable interface is NOT supported in IISS Release 2.0, it will be supported in later releases.

This manual is intended for application and system programmers working in the IISS environment.

SECTION 2

DOCUMENTS

2.1 Reference Documents

[1] Systran, ICAM Documentation Standards, IDS 150120000C, 15 September 1983.

[2] Digital, VAX-11 Architecture Handbook, Digital Equipment Corp., Maynard, MA, 1979.

[3] American National Standards Institute, Code for Information Interchange, ANSI X3.4-1977, 9 June 1977.

[4] American National Standards Institute, Code Extension Techniques for Use with the 7-bit Coded Character Set of American National Standard Code for Information Interchange, ANSI X3.41-1974, 14 May 1974.

[5] American National Standards Institute, Additional Controls for Use With American National Standard Code for Information Interchange, ANSI X3.64-1979, 18 July 1979.

[6] American National Standards Institute, Hollerith Punched Card Code, ANSI X3.26-1980, 2 May 1980.

[7] Structural Dynamics Research Corporation, Report Writer User Manual, UM 20144501 , 1 November 1985.

[8] Structural Dynamics Research Corporation, Application Generator User Manual, UM 620144502 , 1 November 1985.

[9] Structural Dynamics Research Corporation, Text Editor User Manual, UM 620144600B, 1 November 1985.

[10] Structural Dynamics Research Corporation, Form Processor User Manual, UM 620144200B, 1 November 1985.

[11] Structural Dynamics Research Corporation, Form Editor User Manual, UM 620144400B, 1 November 1985.

[12] Structural Dynamics Research Corporation, Virtual Terminal Development Specification, DS 620144300B, 1

November 1985.

2.2 Terms and Abbreviations

American Standard Code for Information Interchange: (ASCII), the character set defined by ANSI X3.4 and used by most computer vendors.

Application Interface: (AI), subset of the IISS User interface that consists of the callable routines that are linked with applications that use the Form Processor or Virtual Terminal. The AI enables applications to be hosted on computers other than the host of the User Interfacce.

Application Process: (AP), a cohesive unit of software that can be initiated as a unit to perform some function or functions.

Attribute: field characteristic such as blinking, highlighted, black, etc. and various other combinatlions. Background attributes are defined for forms or windows only. Foreground attributes are defined for items. Attributes may be permanent, i.e., they remain the same unless changed by the application program, or they may be temporary, i.e., they remain in effect until the window is redisplayed.

Communication Services: allows on host interprocess communicatlion and inter-host communication between the various Test Bed subsystems.

Computer Program Configuration Item: (CPCI), an aggregation of computer programs or any of their discrete portions, which satisfies an end-use function and is designed by the ICAM Program Office for ICAM Configuration Management.

Device Drivers: (DD), software modules written to handle I/O for a specific kind of terminal. The modules map terminal specific commands and data to a neutral format. Device drivers are part of the UI Virtual Terminal.

Extended Binary Coded Decimal Interchange Code: (EBCDIC), the character set used by a few computer vendors (notable IBM) instead of ASCII.

Field: two-dimensional space on a terminal screen.

Integrated Information Support System: (IISS), a test computing environment used to investigate, demonstate and test the concepts of information management and information integration in the context of Aerospace Manufacturing. The IISS addresses the problems of integration of data resident on heterogeneous data bases supported by heterogeneous computers interconnected via a Local Area Network.

Logical Device: a conceptual device which, to an application, is indistinguishable from a physical device and is then mapped to part or all of a physical device.

Network Transaction Manager: (NTM), IISS subsystem that performs the coordination, communication and housekeeping functions required to integrate the Application Processes and System Services resident on the various hosts into a cohesive system.

Operating System: (OS), softawre supplied with a computer which allows it to supervise its own operations and manage access to hardware facilities such as memory and peripherals.

Physical Device: a hardware terminal.

User Interface: (UI), IISS subsystem that controls the user's terminal and interfaces with the rest of the system. The UI consists of two major subsystems: The User Interface Development System (UIDS) and the User Interface Management System (UIMS).

User Interface Management System: (UIMS), the run time UI. It consists of the Form Processor, Virtual Terminal, Application Interface and the User Interface Services.

User Interface Monitor: (UIM), part of the Form Processor that handles messaging between the NTM and the UI. It also provides authoriztion checks and initiates applications.

User Interface/Virtual Terminal Interface: (UI/VTI), another name for the User Interface.

Virtual Terminal: (VT), subset of the IISS User Interface that performs the interfacing between different terminals and the UI. This is done by defining a specific set of terminal features and protocols which must be supported by the UI software which constitutes the virtual terminal definition. Specific terminals are then mapped against the virtual terminal software by specific software modules written for each type of real terminal supported.

Virtual Terminal Interface: (VTI), the callable interface to the VT.

Window: dynamic area of a form in which predefined forms may be placed at runtime.

Window Manager: a facility which allows the following to be manipulated: size and location of windows, the device on which an application is running, the position of a form within a window. It is part of the Form Processor.

SECTION 3

VIRTUAL TERMINAL COMMANDS

3.1 General

The Virtual Terminal accepts two kinds of data: Graphic (or printable) Characters which are displayed on the screen, and Commands which affect the way in which Graphic Characters are displayed.

The format of the following command descriptions is: the command name and short description, the command syntax, and a detailed description of the command. In the command syntax, characters within angle brackets (e.g. ‹ESC›) indicate Control Characters (codings depend on your system character set - see Appendix A), Pn indicates a Numeric Parameter, Ps indicates a Selective Parameter, an ellipsis (...) indicates additional unspecified characters, and all other characters stand for themselves.

Parameters are represented in ordinary human-readable decimal form, with Numeric Parameters representing numbers (such as a row number or the number of times to repeat a function), and Selective Parameters standing for selections from a list of options with multiple selections separated by semicolons. Unless specified otherwise, Numeric Parameters indicate the number of times to repeat the specified function, omitted Numeric Parameters are taken to be 1, and omitted Selective Parameters are taken to be 0.

The Virtual Terminal screen consists of an arbitrary number of rows numbered from 1 to n, and an arbitrary number of columns numbered from 1 to m; the actual size is specified by the Define Window command. The standard ordering of objects is from top to bottom and left to right, with wrap-around from the last object to the first. In the command descriptions, "next" refers to this order, "previous" to its reverse. For example, from row 6 column 80 on an 80 character wide screen, the next character position is row 7 column 1, and the previous character position is row 6 column 79.

In Forms Mode, any command whose effect is limited to a single field (including Graphic Characters) will cause the cursor to move to the next unprotected field before the command takes effect if the cursor is in a protected field when the

command is received. If there are no unprotected fields defined, the command is ignored.

An application program is only permitted to use the following commands: Bell, Define Field, Erase Field, Record Separator, Set Transmit State. The following commands may also be used, subject to constraints: Define Window (window id not specified), Erase Window (window id not specified), all cursor positioning commands (position within logical device bounds). The following commands are for internal use only and may not be used under any circumstances: Define Window (window id specified), Remove Window, Erase Window (window id specified), Set Window, Window Precedence. All other commands may be used, but there is no guarantee that the application will correctly be constrained to the limits of its logical device.

3.2 Command Descriptions

Graphic Character

Causes the character to be displayed according to the graphic rendition in effect at the cursor location and advances the cursor to the next character position. This advancing may possibly causing scrolling.

BEL - Sound Bell

‹BEL›

Sounds an audible alarm at the terminal.

BS - Backspace

‹BS›

Moves the cursor to the previous character position; if the cursor is at the left margin, no action occurs.

HT - Horizontal Tab

‹HT›

Moves the cursor to the next horizontal tab stop on the current line or to the right margin if no more tab stops exist; in Forms Mode, moves the cursor to the next field.

LF - Line Feed

‹LF›

Moves the cursor down to the next line in the current column, possibly scrolling the screen.

FF - Form Feed
‹FF›
Clears the screen and moves the cursor to the first unprotected character position. In Forms Mode, only unprotected areas of the screen are erased.

CR - Carriage Return
‹CR›
Moves the cursor to the left margin in the current line.

RS - Record Separator
‹RS›
Used to indicate the end of a series of commands causing them to be processed and the results displayed.

IND - Index
‹ESC› D
Same as LF.

NEL - Next Line
‹ESC› E
Same as CR followed by LF.

HTS - Horizontal Tab Set
‹ESC› H
Sets a horizontal tab stop at the current column.

RI - Reverse Index
‹ESC› M
Moves the cursor up to the previous line in the current column, possibly scrolling the screen.

DCS - Device Control String
‹ESC› P ... ‹ESC› \
Transmits the characters between the escape sequences (...) directly to the physical terminal without interpretation. This may be used to activate special features of a particular terminal, but it is the user's responsibility to insure that the physical terminal is of the correct type.

STS - Set Transmit State
‹ESC› S
Indicates that the currently selected window is to be enabled for input. All unguarded fields are made enterable and a data message will be sent when a function key is pressed.

APC - Application Program Command
‹ESC› _ Pn ‹ESC› \
Generated when a function key is pressed. The parameter is the function key number (0 - n) which must not be omitted. Function key zero is the "ENTER" key.

RIS - Reset to Initial State
‹ESC› c
Resets the terminal to its initial state. The screen is cleared, the cursor is positioned in the upper left corner, and Forms Mode is reset.

REF - Refresh Screen
‹ESC› ?
Retransmits the current screen contents to the terminal. Its main uses are to recover from unsolicited messages or line noise which have corrupted the screen contents, or to update the terminal when in Deferred Display Mode.

ICH - Insert Character
‹ESC› [Pn @
Makes room for a character by shifting the rest of the line (field in Forms Mode) one character position to the right; characters shifted past the end of the line (field) are lost. The cursor is left at the first inserted character position (i.e. not moved).

CUU - Cursor Up
‹ESC› [Pn A
Moves the cursor to the previous line in the current column, but not past the top margin.

CUD - Cursor Down
‹ESC› [Pn B
Moves the cursor to the next line in the current column, but not past the bottom margin.

CUF - Cursor Forward
‹ESC› [Pn C
Moves the cursor to the next character position, but not past the right margin.

CUB - Cursor Backward
‹ESC› [Pn D
Moves the cursor to the previous character position, but not past the left margin.

CNL - Cursor Next Line
‹ESC› [Pn E
Moves the cursor to the left margin of the next line, but not past the bottom margin.

CPL - Cursor Previous Line
‹ESC› [Pn F
Moves the cursor to the left margin of the previous line, but not past the top margin.

CUP - Cursor Position
‹ESC› [Pn ; Pn H
Moves the cursor to the specified position. The first parameter is the row number, the second parameter is the column number. If the second parameter is omitted, the semicolon may be omitted as well

CHT - Cursor Horizontal Tab
‹ESC› [Pn I
Moves the cursor to the next horizontal tab stop on the current line or the right margin if no more horizontal tab stops exist. In Forms Mode, moves the cursor to the next field.

ED - Erase Display
‹ESC› [Ps J
Erases the screen according to the parameter:
0 - Erase from the cursor to the end of the screen (inclusive)
1 - Erase from the beginning of the screen to the cursor (inclusive)
2 - Erase the entire screen
The cursor is not moved. In Forms Mode, only unprotected areas of the screen are erased.

EL - Erase Line
‹ESC› [Ps K
Erases the current line according to the parameter:
0 - Erase from the cursor to the end of the line (inclusive)
1 - Erase from the beginning of the line to the cursor (inclusive)
2 - Erase the entire line
The cursor is not moved. In Forms Mode, only unprotected areas of the screen are erased.

IL - Insert Line
    ‹ESC› [Pn L
Makes room for a line by shifting the rest of the screen down one line; lines shifted past the bottom of the screen are lost. The cursor is positioned at the first inserted line (i.e. not moved).

DL - Delete Line
    ‹ESC› [Pn M
Deletes the current line by shifting the rest of the screen up one line.

EF - Erase Field
    ‹ESC› [Ps N
Erases the current field according to the parameter:
    0 - Erase from the cursor to the end of the field (inclusive)
    1 - Erase from the beginning of the field to the cursor (inclusive)
    2 - Erase the entire field
The cursor is not moved.

DCH - Delete Character
    ‹ESC› [Pn P
Deletes the current character by shifting the rest of the line (field in Forms Mode) one character position to the left.

CPR - Cursor Position Report
    ‹ESC› [Pn ; Pn R
Generated in reply to a cursor position request (see DSR). The first parameter is the current row, the second parameter is the current column.

NP - Next Page
    ‹ESC› [Pn U
Same as FF.

PP - Previous Page
    ‹ESC› [Pn V
Same as FF.

ECH - Erase Character
    ‹ESC› [Pn X
Erases the current character (the character is NOT deleted). The cursor is not moved. In Forms Mode, only a single field is affected.

CBT - Cursor Backward Tab
‹ESC› [Pn Z
Moves the cursor to the previous horizontal tab stop in the current line or to the left margin if no more horizontal tab stops exist. In Forms Mode, moves the cursor to the previous field.

HPA - Horizontal Position Absolute
‹ESC› [Pn `
Moves the cursor to the specified column in the current line.

HPR - Horizontal Position Relative
‹ESC› [Pn a
Same as CUF.

VPA - Vertical Position Absolute
‹ESC› [Pn d
Moves the cursor to the specified line in the current column.

VPR - Vertical Position Relative
‹ESC› [Pn e
Same as CUD.

HVP - Horizontal and Vertical Position
‹ESC› [Pn ; Pn f
Same as CUP.

TBC - Tab Clear
‹ESC› [Ps g
Clears tab stops according to the parameter:
0 - Clear the horizontal tab stop at the cursor
3 - Clear all horizontal tab stops

SM - Set Mode
‹ESC› [Ps h (standard modes)
‹ESC› [? Ps h (private modes)
Sets the indicated modes; standard and private modes can not be mixed. No standard modes are currently supported. Allowable private mode parameters are:

1 - FRMM - Forms Mode - When set, area qualifications are enforced and reading the terminal results in a full-screen formatted buffer; when reset, area qualifications are not enforced and reads return a single line or command at a time.

3 - CTM - Control Transfer Mode - When set, indicates that control sequences are to be returned to the program; when reset, control sequences terminate a read but are not returned. (Only effective when not in Forms Mode.)

4 - DDM - Deferred Display Mode - When set, indicates that writes are to affect only the internal buffer, not the screen (a REF command should be sent to update the screen); when reset, indicates that writes affect both the internal buffer and the screen.

MC - Media Copy
‹ESC› [Ps i
Controls the transfer of data between the device and an auxiliary input/output device:
0 - Print Screen

RM - Reset Mode
‹ESC› [Ps l (standard modes)
‹ESC› [? Ps l (private modes)
Resets the indicated modes (see Set Mode); standard and private modes can not be mixed.

SGR - Set Graphic Rendition
‹ESC› [Ps m
Sets the specified Graphic Rendition:
0 - Normal (reset existing attributes)
1 - Bright or Bold
2 - Dim
4 - Underlined
5 - Slow Blink (less than 150 per minute)
6 - Fast Blink
7 - Reverse
8 - Concealed (not displayed)

The specified attributes are in effect from the cursor position to the next SGR or the end of the current line, whichever comes first. Note that the specified attributes are IN ADDITION to the currently existing attributes unless Normal is specified.

DSR - Device Status Request
‹ESC› [Ps n
Requests the indicated status:
6 - Report Cursor Position (via CPR)

DAQ - Define Area Qualification
‹ESC› [Ps o
Sets the specified Area Qualification:
0 - No Qualification (reset existing qualifications)
1 - Protected and Guarded
7 - Beginning of Field

The specified qualification is in effect from the cursor position to the next DAQ or the end of the current line, whichever comes first. Note that the specified qualifications are IN ADDITION to the currently existing qualifications unless No Qualification is specified. Area qualifications are only enforced in Forms Mode. DAQ commands take up a single character space on the screen which is displayed as a blank; the cursor is moved to the next character position following a DAQ command. When DAQ and SGR are used together, the SGR command should be given first, followed by the DAQ command. (The screen is completely protected and guarded unless other qualifications are explicitly specified.)

WP - Window Precedence
‹ESC› [Pn ... p
Sets the precedence of the specified windows. Each window is in turn placed on top of all other existing windows. Thus, the last window specified will ultimately be the top-most and all specified windows will be on top of any unspecified windows.

RW - Remove Window
‹ESC› [Pn r
Removes the specified window. If the window id is omitted, the currently selected window is used.

SW - Select Window
‹ESC› [Pn s
Selects the specified window.

EW - Erase Window
‹ESC› [Pn u
Removes all windows and fields from the specified window. If the window id is omitted, the currently selected window is used.

DW - Define Window

‹ESC› [Pn ; Pn ; Pn ; Pn ; Pn ; Pn ; Pn ; Pn ; Ps w

Defines a window within the currently selected window. The first parameter is the window id, the second and third parameters are the row and column within the selected window for this window to be displayed, the fourth and fifth parameters are the display width and depth, the sixth and seventh parameters are the offsets of the first displayed row and column from the actual first row and column, the eighth and ninth parameters are the actual width and depth, and the tenth parameter is the window attributes as per Set Graphic Rendition.

DF - Define Field

‹ESC› [Pn ; Pn ; Pn ; Pn ; Ps ; Ps ; x

Defines a field within the currently selected window. The first and second parameters are the row and column within the selected window for the field to be displayed, the third and fourth parameters are the field width and depth, the fifth parameter is the "guarded" flag which must consist of exactly one selection (unless it and all following parameters are omitted) as per Define Area Qualification (Beginning of Field is implied and should not be specified), and the sixth parameter is the field attributes as per Set Graphic Rendition. The data to be displayed in the field must immediately follow the Define Field command in the same buffer (see PUTVTI, below).

3.3 Input-Output Routines

Four routines are provided for direct Virtual Terminal input and output. The calling sequences and parameter definitions follow.

INITVT

```
CALL "INITVT".
```

This routine performs all necessary initialization in preparation for using the Virtual Terminal. Specifically, it initiates Form Processor Bypass mode wherein the Form Processor no longer interprets Virtual Terminal messages but simply passes them back to the application.

GETVTI

CALL "GETVTI" USING BUFFER, MAX-LEN, ACT-LEN.

Inputs

MAX-LEN - PIC S9(5) COMP - maximum length to read.

Outputs

BUFFER - PIC X(N) - data read from terminal.
ACT-LEN - PIC S9(5) COMP - length of data read.

This routine performs a read from the Virtual Terminal.

In forms mode, the returned buffer consists of a Set Window command followed by Define Field commands for each field in the window which has been modified since the last read. This is followed by additional Set Widow and Define Field commands for nested windows. Finally, a Cursor Position Report command giving the cursor position when the terminating function key was pressed and an Application Program Command command specifying which function key was pressed terminate the buffer.

If not in Forms Mode, the returned buffer consists of all the printable characters entered followed (if in Control Transfer Mode) by the control sequence which terminated the input.

If an inquiry (e.g. DSR) was performed prior to reading, the returned buffer contains only the reply reguardless of Forms Mode and Control Transfer Mode.

PUTVTI

CALL "PUTVTI" USING BUFFER, ACT-LEN.

Inputs

BUFFER - PIC X(n) - Data to be written.
ACT-LEN - PIC S9(5) COMP - Length of data to write.

This routine performs a write to the Virtual Terminal. This routine may be called multiple times to send multiple buffers of commands to the Virtual Terminal. In any case, the final buffer must end with a Record Separator command in order to process the preceding commands. See above for restrictions on the commands which may be contained in BUFFER.

TERMVT

CALL "TERMVT".

This routine terminates the Virtual Terminal. It terminates Form Processor Bypass mode, causing the Form Processor to once again interpret Virtual Terminal messages and refreshes the screen to eliminate any disruption caused by the Virtual Terminal output.

SECTION 4

TERMINAL IMPLEMENTATION

4.1 Adding New Terminals

The translation from Virtual Terminal commands to commands for a specific terminal (and vice versa) is performed by a program known as a device driver. Adding a new terminal is accomplished simply by writing a device driver for the terminal and making it known to the system. Since all device drivers perform the same basic functions, most of the necessary routines are already written, and only a few will need to be written for a particular terminal. (Since the currently existing device drivers are written in the C programming language, a large number of utility and support functions exist for device drivers written in C. For this reason, this discussion will focus on device drivers which are being written in C; this should not be interpreted as meaning that device drivers could not be written in another language, only that doing so would be significantly more work.)

Two different types of device drivers will be discussed. First, we will consider a general purpose device driver which can support any type of terminal. Second, we will consider the special case of a terminal which does not support forms and does not perform local echoing (or allows local echoing to be disabled). It should be noted that all of the currently supported terminals fall into this category.

4.2 General Purpose Device Driver

A general purpose device driver must contain four routines: INITVT, GETVTI, PUTVTI, and TERMVT.

GETVTI and PUTVTI (which have already been discussed) accept Virtual Terminal commands and translate them into commands for a particular device and vice versa. All Virtual Terminal commands must be supported, even if this requires simulation in software. (It should be noted, however, that it is not necessary to allow all Virtual Terminal commands to be entered by the user. It is up to the implementor to determine a reasonable subset to be supported, but the subset should at least include the cursor movements, forward and backward tab, 20 function keys including the enter key, screen refresh, and

delete character.)

The only allowable exceptions to this are the Bell, Media Copy, and Set Graphic Rendition commands. The Bell and Media Copy commands must be recognized correctly, but need not produce any effect if the terminal does not have an audible alarm or printer. Visual attributes should be simulated as well as possible; some guidelines follow.

If the terminal only has two brightness levels, BOLD should be supported with DIM being the same as NORMAL; if only a single brightness level exists, BOLD, DIM, and NORMAL should all be the same. If the terminal has only a single blink speed, it should be used for both FAST BLINK and SLOW BLINK; if blink is not supported, FAST BLINK and SLOW BLINK may be ignored. If only a single highlight is supported (e.g. reverse video, underline, etc.), it should be used for both REVERSE and UNDERSCORE; if no highlights are supported, both REVERSE and UNDERSCORE should be simulated by a software underscore (blanks in the field are replaced by underscores). CONCEALED may be simulated by blanking the field on the screen as necessary.

The Window Manager portion of the Device Driver processes the Set Transmit State, Window Precedence, Define Window, Remove Window, Select Window, Erase Window, and Define Field commands. It is intended to be portable and used in all Device Drivers without change. Thus, these commands do not need to be supported by new Device Drivers. (If, however, the terminal in question supports windowing, it may be desirable to implement these commands as part of the device-specific part of the driver.)

INITVT and TERMVT (which have also been discussed previously) are called once at startup and termination respectively to initialize the device driver and perform cleanup. The initialization usually consists of opening a communication channel to the terminal and calling PUTVTI with a Reset to Initial State command to reset the terminal. The cleanup usually consists of sending commands to the terminal to return it to the normal state of terminals on the system (such as setting normal modes or tab stops) and clear the screen, and closing the communication channel to the terminal.

4.3 Special Case Device Driver

If a terminal supports forms, writing a general purpose device driver for it should not be very difficult. However, a

terminal which does not support forms requires most functions to be simulated in software, requiring a very complex device driver. Since all of the terminals which are currently supported fall into this category, routines exist which make writing a device driver for this type of terminal much easier. (However, it should be noted that supporting this type of terminal requires being able to perform character at a time I/O without echo. This is not possible on some computer systems, making support impossible.) These support routines are documented in Appendix C; many unsubstantiated references to them will be made in the following text.

Supporting a new terminal of this type requires writing six routines: TRMINI, TRMCHK, TRMGET, TRMPUT, TRMFLS, and TRMEND. TRMINI is called once to establish communication with the terminal. This is usually done with a call to TBOPEN. The calling sequence for TRMINI is:

```
trmini(tname)
```

TNAME is the terminal name passed in to INITVT converted to a C string.

TRMCHK is called to check for terminal input that must be processed. The calling sequence for TRMCHK is:

```
trmchk()
```

It returns TRUE or FALSE depending on whether there are keyboard characters to be processed or not.

TRMGET and TRMPUT are called to get commands from and put commands to the terminal. TRMGET usually calls TRMPUT as well in order to echo the user input. The calling sequences for TRMGET and TRMPUT are:

```
trmget(cmd)
trmput(cmd)
```

CMD is a command in internal form.

TRMFLS is called to insure that all output has been displayed (any buffers should be flushed). The calling sequence for TRMFLS is:

```
trmfls()
```

TRMEND is called once to terminate communications with the terminal. The calling sequence for TRNEND is:

```
trmend()
```

APPENDIX A

VIRTUAL TERMINAL CHARACTER SET

	ASCII			EBCDIC				ASCII			EBCDIC		
Char	Hex	Oct	Dec	Hex	Oct	Dec	Char	Hex	Oct	Dec	Hex	Oct	Dec
‹NUL›	00	000	0	00	000	0	‹SP›	20	040	32	40	100	64
‹SOH›	01	001	1	01	001	1	!	21	041	33	4F	117	79
‹STX›	02	002	2	02	002	2	"	22	042	34	7F	177	127
‹ETX›	03	003	3	03	003	3	#	23	043	35	7B	173	123
‹EOT›	04	004	4	37	067	55	$	24	044	36	5B	133	91
‹ENQ›	05	005	5	2D	055	45	%	25	045	37	6C	154	108
‹ACK›	06	006	6	2E	056	46	&	26	046	38	50	120	80
‹BEL›	07	007	7	2F	057	47	'	27	047	39	7D	175	125
‹BS›	08	010	8	16	026	22	(	28	050	40	4D	115	77
‹HT›	09	011	9	05	005	5	)	29	051	41	5D	135	93
‹LF›	0A	012	10	25	045	37	*	2A	052	42	5C	134	92
‹VT›	0B	013	11	0B	013	11	+	2B	053	43	4E	116	78
‹FF›	0C	014	12	0C	014	12	,	2C	054	44	6B	153	107
‹CR›	0D	015	13	0D	015	13	-	2D	055	45	60	140	96
‹SO›	0E	016	14	0E	016	14	.	2E	056	46	4B	113	75
‹SI›	0F	017	15	0F	017	15	/	2F	057	47	61	141	97
‹DLE›	10	020	16	10	020	16	0	30	060	48	F0	360	240
‹DC1›	11	021	17	11	021	17	1	31	061	49	F1	361	241
‹DC2›	12	022	18	12	022	18	2	32	062	50	F2	362	242
‹DC3›	13	023	19	13	023	19	3	33	063	51	F3	363	243
‹DC4›	14	024	20	3C	074	60	4	34	064	52	F4	364	244
‹NAK›	15	025	21	3D	075	61	5	35	065	53	F5	365	245
‹SYN›	16	026	22	32	062	50	6	36	066	54	F6	366	246
‹ETB›	17	027	23	26	046	38	7	37	067	55	F7	367	247
‹CAN›	18	030	24	18	030	24	8	38	070	56	F8	370	248
‹EM›	19	031	25	19	031	25	9	39	071	57	F9	371	249
‹SUB›	1A	032	26	3F	077	63	:	3A	072	58	7A	172	122
‹ESC›	1B	033	27	27	047	39	;	3B	073	59	5E	136	94
‹FS›	1C	034	28	1C	034	28	<	3C	074	60	4C	114	76
‹GS›	1D	035	29	1D	035	29	=	3D	075	61	7E	176	126
‹RS›	1E	036	30	1E	036	30	>	3E	076	62	6E	156	110
‹US›	1F	037	31	1F	037	31	?	3F	077	63	6F	157	111

Char	ASCII Hex	ASCII Oct	ASCII Dec	EBCDIC Hex	EBCDIC Oct	EBCDIC Dec	Char	ASCII Hex	ASCII Oct	ASCII Dec	EBCDIC Hex	EBCDIC Oct	EBCDIC Dec
@	40	100	64	7C	174	124	`	60	140	96	79	171	121
A	41	101	65	C1	301	193	a	61	141	97	81	201	129
B	42	102	66	C2	302	194	b	62	142	98	82	202	130
C	43	103	67	C3	303	195	c	63	143	99	83	203	131
D	44	104	68	C4	304	196	d	64	144	100	84	204	132
E	45	105	69	C5	305	197	e	65	145	101	85	205	133
F	46	106	70	C6	306	198	f	66	146	102	86	206	134
G	47	107	71	C7	307	199	g	67	147	103	87	207	135
H	48	110	72	C8	310	200	h	68	150	104	88	210	136
I	49	111	73	C9	311	201	i	69	151	105	89	211	137
J	4A	112	74	D1	321	209	j	6A	152	106	91	221	145
K	4B	113	75	D2	322	210	k	6B	153	107	92	222	146
L	4C	114	76	D3	323	211	l	6C	154	108	93	223	147
M	4D	115	77	D4	324	212	m	6D	155	109	94	224	148
N	4E	116	78	D5	325	213	n	6E	156	110	95	225	149
O	4F	117	79	D6	326	214	o	6F	157	111	96	226	150
P	50	120	80	D7	327	215	p	70	160	112	97	227	151
Q	51	121	81	D8	330	216	q	71	161	113	98	230	152
R	52	122	82	D9	331	217	r	72	162	114	99	231	153
S	53	123	83	E2	342	226	s	73	163	115	A2	242	162
T	54	124	84	E3	343	227	t	74	164	116	A3	243	163
U	55	125	85	E4	344	228	u	75	165	117	A4	244	164
V	56	126	86	E5	345	229	v	76	166	118	A5	245	165
W	57	127	87	E6	346	230	w	77	167	119	A6	246	166
X	58	130	88	E7	347	231	x	78	170	120	A7	247	167
Y	59	131	89	E8	350	232	y	79	171	121	A8	250	168
Z	5A	132	90	E9	351	233	z	7A	172	122	A9	251	169
[	5B	133	91	4A	212	74	{	7B	173	123	C0	300	192
\	5C	134	92	E0	340	224	|	7C	174	124	6A	152	106
]	5D	135	93	5A	132	90	}	7D	175	125	D0	320	208
^	5E	136	94	5F	137	95	~	7E	176	126	A1	241	161
_	5F	137	95	6D	155	109	<DEL>	7F	177	127	07	007	7

APPENDIX B

COMMAND REFERENCE

For each function the key sequence, internal function identifier, command abbreviation, and command description are given. Tables of selective parameters follow the function definitions.

Function Definitions

Control Characters

Ctrl-G	0007	BEL	Sound Bell
Ctrl-H	0008	BS	Backspace
Ctrl-I	0009	HT	Forward Tab
Ctrl-J	0010	LF	Line Feed / New Line
Ctrl-L	0012	FF	Form Feed
Ctrl-M	0013	CR	Carriage Return
Ctrl-[		ESC	Character Set Extension (see following)
Ctrl-^	0030	RS	Record Separator
‹ESC›D	1004	IND	Index
‹ESC›E	1005	NEL	Next Line
‹ESC›H	1008	HTS	Horizontal Tab Set
‹ESC›M	1013	RI	Reverse Index
‹ESC›P	1016	DCS	Device Control String
‹ESC›S	1019	STS	Set Transmit State
‹ESC›[		CSI	Control Sequence Introducer (see following)
‹ESC›\	1028	ST	String Terminator
‹ESC›_	1031	APC	Application Program Command (function keys)
‹ESC›c	1035	RIS	Reset to Initial State
‹ESC›?	4000	REF	Refresh Screen (private)

Control Sequences (‹CSI› . . .)

Pn @	3000	ICH	Insert Character
Pn A	3001	CUU	Cursor Up
Pn B	3002	CUD	Cursor Down
Pn C	3003	CUF	Cursor Forward
Pn D	3004	CUB	Cursor Backward
Pn E	3005	CNL	Cursor Next Line
Pn F	3006	CPL	Cursor Preceding Line
Pn;Pn H	3008	CUP	Cursor Position
Pn I	3009	CHT	Cursor Horizontal Tab
Ps J	3010	ED	Erase Display
Ps K	3011	EL	Erase Line

Pn L	3012	IL	Insert Line
Pn M	3013	DL	Delete Line
Ps N	3014	EF	Erase Field
Pn P	3016	DCH	Delete Character
Pn;Pn R	3018	CPR	Cursor Position Report
Pn U	3021	NP	Next Page
Pn V	3022	PP	Preceding Page
Pn X	3024	ECH	Erase Character
Pn Z	3026	CBT	Cursor Backward Tab
Pn `	3032	HPA	Horizontal Position Absolute
Pn a	3033	HPR	Horizontal Position Relative
Pn d	3036	VPA	Vertical Position Absolute
Pn e	3037	VPR	Vertical Position Relative
Pn.Pn f	3038	HVP	Horizontal and Vertical Position
Ps g	3039	TBC	Tab Clear
Ps h	3040	SM	Set Mode
Ps i	3041	MC	Media Copy
Ps l	3044	RM	Reset Mode
Ps m	3045	SGR	Set Graphic Rendition
Ps n	3046	DSR	Device Status Request
Ps o	3046	DAQ	Define Area Qualification
Pn ... p	3047	WP	Window Precedence
Pn r	3049	RW	Remove Window
Pn s	3050	SW	Set Window
Pn u	3052	EW	Erase Window
Pn w	3054	DW	Define Window
Pn x	3055	DF	Define Field
? Ps h	4040	SPM	Set Private Mode (private)
? Ps l	4044	RPM	Reset Private Mode (private)

Device Control Strings (DCS ... ST)

Characters to be sent to terminal without interpretation

Application Program Commands (APC ... ST)

Decimal representation of function key number

Selective Parameter Tables

Erase Parameters

0 - Current Position to End of Area (inclusive)
1 - Beginning of Area to Current Position (inclusive)
2 - Entire Area

Tab Clear Parameters

0 - Horizontal Tab at Current Position
3 - All Horizontal Tabs

Mode Parameters

none

Private Mode Parameters

1 - FRMM Form Mode
3 - CTM Control Transfer Mode
4 - DDM Deferred Display Mode

Media Copy Parameters

0 - Print Screen

Graphic Rendition Parameters

0 - Default
1 - Bright
2 - Dim
4 - Underscore
5 - Slow Blink
6 - Fast Blink
7 - Reverse
8 - Concealed

Device Status Request Parameters

6 - Report Current Position (via CPR)

Area Qualification Parameters

0 - No Qualification
1 - Guarded
7 - Set Tab Stop (field delimiter)

APPENDIX C

DEVICE DRIVER SUPPORT ROUTINES

TERMIO.H

```
/* NAME
 *    termio - terminal i/o package
 *       Written: 11-MAY-1983 15:54:05
 *       Revised: 13-SEP-1983 12:24:19
 *
 * DESCRIPTION
 *    This package provides immediate, character at a time i/o
 *    from a terminal (i.e. does not collect an edited line
 *    like stdio).
 *
 *    For details on the supported functions, see the
 *    individual function descriptions.
 */

/* NAME
 *    topen - open terminal channel
 *
 * SYNOPSIS
 *    #include <termio.h>
 *
 *    TERM *topen(device)
 *       char *device;
 *
 * DESCRIPTION
 *    topen opens the terminal specified by device for terminal
 *    i/o
 *
 *    device is a pointer to a string containing the device
 *    name.
 *
```

```
/* NAME
 *    tbopen - open buffered terminal channel
 *
 * SYNOPSIS
 *    #include <termio.h>
 *
 *    TERM *tbopen(device, bufsiz, nbuf)
 *       char *device;
 *       int bufsiz, nbuf; *
 * DESCRIPTION
 *    tbopen opens the terminal specified by device for
 *    buffered terminal i/o.
 *    (Only the output is buffered, not the input.)
 *
 *    device is a pointer to a string containing the device
 *    name.
 *    bufsiz is the buffer size in characters.
 *    nbuf is the number of buffers to allocate.
 *
 *    If nbuf (or bufsiz) is zero, the terminal is opened
 *    unbuffered.
 */

/* NAME
 *    tgetc - get character
 *
 * SYNOPSIS
 *    #include <termio.h>
 *
 *    char tgetc(term)
 *       TERM *term;
 *
 * DESCRIPTION
 *    tgetc returns the next character typed at the specified
 *    terminal.
 */
```

```
/* NAME
 *    tgetct - get character (transparent)
 *
 * SYNOPSIS
 *    #include <termio.h>
 *
 *    char tgetct(term)
 *       TERM *term;
 *
 * DESCRIPTION
 *    tgetct returns the next character typed at the specified
 *    terminal without processing special control characters.
 *    Note that characters already in the type-ahead buffer may
 *    have been subject to special processing.
 */

/* NAME
 *    tungetc - unget character
 *
 * SYNOPSIS
 *    #include <termio.h>
 *
 *    char tgetct(c, term)
 *       char c;
 *       TERM *term;
 *
 * DESCRIPTION
 *    tungetc returns the specified character to the specified
 *    terminal so that the next tgetc or tgetct call will
 *    return it. Only a single push-back is allowed.
 */

/* NAME
 *    tputc - put character
 *
 * SYNOPSIS
 *    #include <termio.h>
 *
 *    void tputc(c, term)
 *       char c;
 *       TERM *term;
 *
 * DESCRIPTION
 *    tputc outputs the specified character to the specified
 *    terminal.
 */
```

```
/* NAME
 *    tputct - put character (transparent)
 *
 * SYNOPSIS
 *    #include <termio.h>
 *
 *    void tputct(c, term)
 *       char c;
 *       TERM *term;
 *
 * DESCRIPTION
 *    tputct outputs the specified character to the specified
 *    terminal without processing special control characters.
 */

/* NAME
 *    tflush - flush terminal buffer
 *
 * SYNOPSIS
 *    #include <termio.h>
 *
 *    void tflush(term)
 *       TERM *term;
 *
 * DESCRIPTION
 *    tflush empties the specified terminal's output buffer.
 */

/* NAME
 *    tflusht - flush terminal buffer (transparent)
 *
 * SYNOPSIS
 *    #include <termio.h>
 *
 *    void tflusht(term)
 *       TERM *term;
 *
 * DESCRIPTION
 *    tflusht empties the specified terminal's output buffer
 *    without interpreting special control characters.
 */
```

```
/* NAME
 *    tclose - close terminal
 *
 * SYNOPSIS
 *    #include <termio.h>
 *
 *    void tclose(term)
 *       TERM *term;
 *
 * DESCRIPTION
 *    tclose closes the specified terminal.
 */

/* NAME
 *    ttrans - set transparent mode
 *
 * SYNOPSIS
 *    #include <termio.h>
 *
 *    void ttrans(term)
 *       TERM *term;
 *
 * DESCRIPTION
 *    ttrans places the terminal in transparent mode.  In this
 *    mode, all special characters (ctrl-y, ctrl-c, ctrl-s,
 *    ctrl-q, ctrl-o, ctrl-r, and ctrl-t) are treated as data
 *    and returned by tgetc.
 */

/* NAME
 *    tntrans - reset transparent mode
 *
 * SYNOPSIS
 *    #include <termio.h>
 *
 *    void tntrans(term)
 *       TERM *term;
 *
 * DESCRIPTION
 *    tntrans cancels transparent mode set by ttrans.
 */
```

```
/* NAME
 *    tcheck - check for input
 *
 * SYNOPSIS
 *    #include <termio.h>
 *
 *    void tcheck(term)
 *       TERM *term;
 *
 * DESCRIPTION
 *    tcheck returns the number of characters in the type-ahead
 *    buffer.
 */

/* NAME
 *    tpurge - purge typeahead
 *
 * SYNOPSIS
 *    #include <termio.h>
 *
 *    void tpurge(term)
 *       TERM *term;
 *
 * DESCRIPTION
 *    tpurge removes all characters from the typeahead buffer.
 */

/* NAME
 *    tgetnm - get device name
 *
 * SYNOPSIS
 *    #include <termio.h>
 *
 *    char *tgetnm(dev)
 *       char *dev;
 *
 * DESCRIPTION
 *    returns the physical device name associated with the
 *    specified logical device name
 */
```

TPUTS.C

```
/* NAME
 *    tputs - Terminal PUT String
 *       Written:  3-JUN-1983 10:14:03
 *       Revised: 24-AUG-1983 09:43:27
 *
 * SYNOPSIS
 *    void tputs(s, chan)
 *       char *s;
 *       TERM *chan;
 *
 * DESCRIPTION
 *    Writes the specified string to the specified terminal.
 */
```

DOSCREEN.C

```
/* NAME
 *    doscreen - DO command to internal SCREEN
 *       Written: 25-MAY-1983 09:53:08
 *       Revised: 13-SEP-1983 10:40:48
 *
 * SYNOPSIS
 *    #include "screen.h"
 *
 *    int doscreen(cmd)
 *       struct command *cmd;
 *
 * DESCRIPTION
 *    Executes cmd on the internal screen and fixes up its
 *    parameters.
 *    Returns -1 for errors, 0 for no action, 1 for normal
 *    command, and 2 for move the cursor and retry.
 */
```

APPENDIX D

DEVICE DRIVER INCLUDE FILES

SCREEN.H

```
/* NAME
 *     screen.h - internal SCREEN definitions
 *        Written: 19-MAY-1983 14:18:12
 *        Revised: 10-JAN-1985 07:05:24 - SCWEHRMAN
 *
 * DESCRIPTION
 *   Defines symbols, externals, etc. for the internal screen
 *   buffer.
 */

#ifndef POS

#define HTABSIZ maxx
#define DSRSIZ  7

#define POS(y, x) ((x)+(y)*maxx)
#define ROW(pos) ((pos)/maxx)
#define COL(pos) ((pos)%maxx)

extern int maxx, maxy, curpos, maxpos, chgmax, chgmin;

extern char *screen;

struct attr {
#define FLAG 0
   bits gr;
#define BOLD 1
#define DIM  2
#define UNDR 4
#define SLBL 5
#define FABL 6
#define REV  7
#define NDSP 8
#define GRSIZ 9
   bits aq;
#define PROT 1
#define HTAB 7
#define MDFY 15
#define AQSIZ 16
   };
extern struct attr *attrib;
```

```
extern struct attr DEFATR;

extern BITTYP *htab;

struct modes {
#define GATM  1
#define KAM   2
#define CRM   3
#define IRM   4
#define SRTM  5
#define ERM   6
#define VEM   7
#define HEM  10
#define PUM  11
#define SRM  12
#define FEAM 13
#define FETM 14
#define MATM 15
#define TTM  16
#define SATM 17
#define TSM  18
#define EBM  19
#define LNM  20
#define NSMODE 21
   BITSTR(smode, NSMODE);
#define FRMM  1
#define FESM  2
#define CTM   3
#define DDM   4
#define NPMODE 5
   BITSTR(pmode, NPMODE);
   };
extern struct modes vti;

extern bool xmit;

#endif
```

```
/* NAME
 *    functs.h - FUNCTion definitionS
 *       Written: 24-AUG-1983 09:49:37
 *       Revised:  9-MAY-1985 10:31:48 - WEHRMAN
 *
 * DESCRIPTION
 *    Defines the mnemonic virtual terminal command functions.
 *    And defines structure for parsing vti message buffer.
 */
#ifndef FUNCFLAG
#define FUNCFLAG 1

#define BEL 7
#define BS  8
#define HT  9
#define LF  10
#define FF  12
#define CR  13
#define RS  30
#define US  31
#define IND 1004
#define NEL 1005
#define HTS 1008
#define RI  1013
#define DCS 1016
#define STS 1019
#define ST  1028
#define APC 1031
#define RIS 1035
#define ICH 3000
#define CUU 3001
#define CUD 3002
#define CUF 3003
#define CUB 3004
#define CNL 3005
#define CPL 3006
#define CUP 3008
#define CHT 3009
#define ED  3010
#define EL  3011
#define IL  3012
#define DL  3013
#define EF  3014
#define DCH 3016
#define CPR 3018
#define NP  3021
#define PP  3022
```

```
#define ECH 3024
#define CBT 3026
#define HPA 3032
#define HPR 3033
#define VPA 3036
#define VPR 3037
#define HVP 3038
#define TBC 3039
#define SM  3040
#define MC  3041
#define RM  3044
#define SGR 3045
#define DSR 3046
#define DAQ 3047
#define WP  3048
#define RW  3050
#define SW  3051
#define EW  3053
#define DW  3055
#define DF  3056
#define REF 4000
#define SPM 4040
#define RPM 4044

typedef struct command
   {
   int funct, maxparm, nparm, parm[];
   } CMD;
#define BLDCMD(n) struct{int funct, maxparm, nparm, parm[n];}

extern bool pass_thru;
#endif
```

APPENDIX E

SAMPLE DEVICE DRIVER (DEC VT-100)

```
#define PRINTER "CI600.C"

#ifndef NDEBUG
#include <stdio.h>
#endif

/* NAME
 *    vt100 - vt100 terminal driver routines
 *       Written: 25-MAY-1983 11:32:20
 *       Revised:  2-AUG-1985 13:41:22 - JONES
 *
 * DESCRIPTION
 *    Device dependent modules for the DEC VT100 device driver.
 */


#include <stdtyp.h>
#include <ctype.h>
#include <termio.h>
#include <bits.h>
#include <screen.h>
#include <functs.h>
#include <trmrtn.h>

#define BUFSIZ 512
#define BUFNUM 2

static TERM *chan;
static int  termpos, pendpos;

static void movcur();
static void setatr();
void trmput();
```

```
/* NAME
 *    trmini - TeRMinal INItialize
 *
 * SYNOPSIS
 *    void trmini(tname)
 *       char *tname;
 *
 * DESCRIPTION
 *    Opens the terminal specified by tname and initializes it.
 */

void trmini(tname)
   char *tname;
   {

   chan = tbopen(tname, BUFSIZ, BUFNUM);
# ifdef PRINTER
   prnini(tname);
# endif
   }
```

```
/* NAME
 *    trmend - TeRMinal END
 *
 * SYNOPSIS
 *    void trmend()
 *
 * DESCRIPTION
 *    Resets the currently open terminal and closes it.
 */

void trmend()
   {
   register int i;

# ifdef PRINTER
   prnend();
# endif
   for (i = 9; i < 80; i += 8) tputs("\33[8C\33H", chan);
   tputs("\33>\r", chan);
   tclose(chan);
   }
```

```
/* NAME
 *    trmfls - TeRMinal FLuSh
 *
 * SYNOPSIS
 *    void trmfls()
 *
 * DESCRIPTION
 *    Flush any terminal buffers.
 */

void trmfls()
   {

   if (pendpos != 0) movcur(pendpos);
   if (!xmit) tflush(chan);
   }
```

```
/* NAME
 *     trmchk - TeRMinal CHecK
 *
 * SYNOPSIS
 *     int trmchk()
 *
 * DESCRIPTION
 *     This module returns the number of characters in the
type-ahead buffer.
 */

int trmchk()
    {

    return tcheck(chan);
    }
```

```
/* NAME
 *    trmget - TeRMinal GET
 *
 * SYNOPSIS
 *    void trmget(cmd)
 *       struct command *cmd;
 *
 * DESCRIPTION
 *    Gets the next command from the terminal and converts it to
 *    internal form.
 */

void trmget(cmd)
   struct command *cmd;
   {
   register char c;
   register int num, i;
   static BLDCMD(2) curcmd = { CUP, 2, 2, 0, 0 };


#ifndef NDEBUG
#     if (termpos != curpos)
         {
         printf("\n  >>  Sync error (trmget): termpos = %d, curpos
         = %d  <<  \n",
               termpos, curpos);
         getchar();
         }
#endif

   if (xmit) tpurge(chan);

   if (isprint(c = tgetc(chan)))        /* printable */
      {
      cmd->funct = 0;
      cmd->nparm = 1;
      cmd->parm[0] = c;
      }
   else if (c != '\33')                    /* control char */
      {
      if (c == '\22'  ||  c == '\27') cmd->funct = REF;
      else if (c == '\177') cmd->funct = DCH;
      else cmd->funct = c;
      cmd->nparm = 0;
      }
```

```
else switch (c = tgetc(chan))
   {
   case 'O':   /* APC */
      cmd->funct = APC;
      cmd->nparm = 1;
      cmd->parm[0] = 1;
      switch (c = tgetc(chan))
         {
         case 'M': cmd->parm[0] = 0; break;
         case 'P': cmd->parm[0] = 1; break;
         case 'Q': cmd->parm[0] = 2; break;
         case 'R': cmd->parm[0] = 3; break;
         case 'S': cmd->parm[0] = 4; break;
         case 'w': cmd->parm[0] = 5; break;
         case 'x': cmd->parm[0] = 6; break;
         case 'y': cmd->parm[0] = 7; break;
         case 'm': cmd->parm[0] = 8; break;
         case 't': cmd->parm[0] = 9; break;
         case 'u': cmd->parm[0] = 10; break;
         case 'v': cmd->parm[0] = 11; break;
         case 'l': cmd->parm[0] = 12; break;
         case 'q': cmd->parm[0] = 13; break;
         case 'r': cmd->parm[0] = 14; break;
         case 's': cmd->parm[0] = 15; break;
         case 'p': cmd->parm[0] = 16; break;
         case 'n': cmd->parm[0] = 17; break;
         case 'A': cmd->funct = CUU; break;
         case 'B': cmd->funct = CUD; break;
         case 'C': cmd->funct = CUF; break;
         case 'D': cmd->funct = CUB; break;
         }
      break;
   case '\t':                    /* back tab */
      cmd->funct = CBT;
      cmd->nparm = 1;
      cmd->parm[0] = 1;
      break;
   case '\12':                   /* erase end of field */
      cmd->funct = EF;
      cmd->nparm = 0;
      break;
   case '\177':                  /* insert/overstrike mode */
      cmd->funct = tbit(&vti.smode, NSMODE, IRM) ? RM : SM;
      cmd->nparm = 1;
      cmd->parm[0] = IRM;
      break;
```

```
                                    /* function keys */
case '1':  case '2':  case '3':  case '4':  case '5':
           CASE '6':  CASE '7':  CASE '8':  CASE '9':
   cmd->parm[0] = c - '0';
   goto pfcom;
case '0':
   cmd->parm[0] = 10;
   goto pfcom;
case 'q':
   cmd->parm[0] = 11;
   goto pfcom;
case 'w':
   cmd->parm[0] = 12;
   goto pfcom;
case 'e':
   cmd->parm[0] = 13;
   goto pfcom;
case 'r':
   cmd->parm[0] = 14;
   goto pfcom;
case 't':
   cmd->parm[0] = 15;
   goto pfcom;
case 'y':
   cmd->parm[0] = 16;
   goto pfcom;
case 'u':
   cmd->parm[0] = 17;
   goto pfcom;
case 'i':
   cmd->parm[0] = 18;
   goto pfcom;
case 'o':
   cmd->parm[0] = 19;
   goto pfcom;
case 'p':
   cmd->parm[0] = 20;
   goto pfcom;
case '\r':
   cmd->parm[0] = 0;
pfcom:
   cmd->funct = APC;
   cmd->nparm = 1;
   break;
case '[':                          /* control sequence */
   i = 0;
   do
```

```
            {
            num = 0;
            while(isdigit(c = tgetc(chan))) num = 10 * num  +  c
            - '0';
            cmd->parm[i++] = num;
            } while (c == ';');
         cmd->funct = 3000 + c - '@';
         cmd->nparm = i;
         break;
      default:
         cmd->funct = 1000 + c - '@';
         cmd->nparm = 0;
      }
   if ((i = doscreen(cmd)) == 2)
      {
      curcmd.parm[0] = ROW(curpos) + 1;
      curcmd.parm[1] = COL(curpos) + 1;
      trmput(&curcmd);
      doscreen(cmd);
      }
   if (i > 0) trmput(cmd);
   trmfls();
   }
```

```
/* NAME
 *    trmput - TeRMinal PUT
 *
 * SYNOPSIS
 *    void trmput(cmd)
 *       struct command *cmd;
 *
 * DESCRIPTION
 *    Puts an internal format command to the terminal.
 */

#ifndef NDEBUG
static rflag = -1;
#endif

void trmput(cmd)
   struct command *cmd;
   {
   int i, j, k, savepos;
   char c;
   struct attr tnew;
   static struct attr tattr;

#ifndef NDEBUG
   rflag++;
#endif

   switch (cmd->funct)
      {
      case 0:
         if (tbit(&vti.smode, NSMODE, IRM))
            {
            pendpos = curpos;
            goto ref;
            }
         if (pendpos >= 0) movcur(pendpos);
         j = tbit(&attrib[termpos].aq, AQSIZ, FLAG) ?
               1<<NDSP : attrib[termpos].gr;
         if (ffbda(&j, &tattr.gr, GRSIZ, 0) > 0)
            {
            cabit(&tnew.gr, GRSIZ);
            i = 0;
            while ((k = i = ffbsa(&j, GRSIZ, i)) >= 0)
               {
               if (k == FABL) k = SLBL;
               if (k != DIM  &&  k != NDSP) sbit(&tnew.gr,
               GRSIZ, k);
```

```
            }
         if (ffbda(&tnew.gr, &tattr.gr, GRSIZ, -1) > 0)
            setatr(tattr.gr = tnew.gr);
         }
      c = cmd->parm[0];
      if (tbit(&attrib[termpos].gr, GRSIZ, NDSP)  ||  c ==
      '\0') c = ' ';
      tputc(c, chan);
      if (COL(++termpos) == 0)
         {
         pendpos = termpos - 1;
         termpos = -1;
         }
      break;
   case BEL:
      tputc('\7', chan);
      break;
   case BS:
      if (pendpos < 0) pendpos = termpos;
      pendpos--;
      break;
   case NEL:
      if (pendpos < 0) pendpos = termpos;
      pendpos -= COL(pendpos);
   case LF:
   case IND:
      if (pendpos < 0) pendpos = termpos;
      if (ROW(pendpos) < maxy - 1)
         pendpos += maxx;
      else
         {
         movcur(pendpos);
         tputc('\12', chan);
         }
      break;
   case FF:
   case NP:
   case PP:
      movcur(0);
      if (tbit(&vti.pmode, NPMODE, FRMM)) refresh();
      else tputs("\33[J", chan);
      break;
   case CR:
      if (pendpos < 0) pendpos = termpos;
      pendpos -= COL(pendpos);
      break;
   case HTS:
```

```
            if (pendpos >= 0) movcur(pendpos);
            tputs("\33H", chan);
            break;
         case RI:
            if (pendpos < 0) pendpos = termpos;
            if (pendpos >= maxx)
               pendpos -= maxx;
            else
               {
               movcur(pendpos);
               tputs("\33M", chan);
               }
            break;
         case RIS:

tputs("\33<\33[H\33[J\33[?1;3l\33[?7h\33[4;20l\33[3g\33[m\
33=\33[q",
                  chan);       /* removed \33[12h for Tek 410x
                               firmware bug */
            termpos = 0;
            pendpos = -1;
            cabit(&tattr.gr, GRSIZ);
            break;
         case CPL:
            if (pendpos < 0) pendpos = termpos;
            pendpos -= COL(pendpos);
         case CUU:
            if (pendpos < 0) pendpos = termpos;
            pendpos -= cmd->parm[0] * maxx;
            break;
         case CNL:
            if (pendpos < 0) pendpos = termpos;
            pendpos -= COL(pendpos);
         case CUD:
         case VPR:
            if (pendpos < 0) pendpos = termpos;
            pendpos += cmd->parm[0] * maxx;
            break;
         case CUF:
         case HPR:
            if (pendpos < 0) pendpos = termpos;
            pendpos += cmd->parm[0];
            break;
         case CUB:
            if (pendpos < 0) pendpos = termpos;
            pendpos -= cmd->parm[0];
            break;
```

```
case CUP:
case HVP:
case CPR:
   pendpos = POS(cmd->parm[0]-1, cmd->parm[1]-1);
   break;
case HT:
case CHT:
   if (tbit(&vti.smode, NSMODE, TSM))
      pendpos = curpos;
   else
      {
      if (pendpos >= 0) movcur(pendpos);
      for (j = 0; j < cmd->parm[0]; j++)
         tputc('\11', chan);
      termpos = curpos;
      pendpos = -1;
      }
   break;
case ED:
   if (tbit(&vti.pmode, NPMODE, FRMM)) goto ref;
   else
      {
      if (pendpos >= 0) movcur(pendpos);
      tputs("\33[", chan);
      if (cmd->parm[0] > 0) tputnum(cmd->parm[0], chan);
      tputc('J', chan);
      }
   break;
case EL:
   if (tbit(&vti.pmode, NPMODE, FRMM)) goto ref;
   else
      {
      if (pendpos >= 0) movcur(pendpos);
      tputs("\33[", chan);
      if (cmd->parm[0] > 0) tputnum(cmd->parm[0], chan);
      tputc('K', chan);
      }
   break;
case CBT:
case DAQ:
   pendpos = curpos;
   break;
case HPA:
   if (pendpos < 0) pendpos = termpos;
   pendpos += cmd->parm[0]-1 - COL(pendpos);
   break;
```

```
case VPA:
   if (pendpos < 0) pendpos = termpos;
   pendpos += (cmd->parm[0]-1 - ROW(pendpos)) * maxx;
   break;
case TBC:
   if (pendpos >= 0) movcur(pendpos);
   tputs("\33[", chan);
   for (i = 0; i < cmd->nparm; i++)
      {
      if (i > 0) tputc(';', chan);
      if (cmd->parm[i] > 0) tputnum(cmd->parm[i], chan);
      }
   tputc('g', chan);
   break;
case DSR:
   if (pendpos >= 0) movcur(pendpos);
   tputs("\33[6n", chan);
   xmit = 1;
   break;
case REF:
   tputs("\33<\33[?1;3l\33[?7h\33[4;20l\33[m\33=",
         chan);       /* removed \33[12h for Tek 410x
                         firmware bug */
   /* this is really not sufficient - need to reset tabs,
   etc. */
   termpos = pendpos = -1;
   cabit(&tattr.gr, GRSIZ);
   refresh();
   break;
case US:
   pendpos = curpos;
case ICH:
case IL:
case DL:
case EF:
case DCH:
case ECH:
case SGR:
ref:
   if (chgmax > 0)
      {
      savepos = (pendpos >= 0 ? pendpos : termpos);
      refterm(chgmin, chgmax);
      pendpos = savepos;
      }
   break;
```

```
        case SM:
           for (i = 0; i < cmd->nparm; i++)
              if (cmd->parm[i] == IRM) tputs("\33[1q", chan);
           break;
        case RM:
           for (i = 0; i < cmd->nparm; i++)
              if (cmd->parm[i] == IRM) tputs("\33[q", chan);
           break;
        case MC:
# ifdef PRINTER
           cmd->funct = REF;
           prnput(cmd);
           prnfls();
           break;
# endif
        case RS:
        case APC:
        case SPM:
        case RPM:
           break;
        }

#ifndef NDEBUG
    if (!rflag  &&  (pendpos >= 0 ? pendpos : termpos) != curpos)
        {
        printf("\n  >>  Sync error (trmput): termpos = %d, curpos
        = %d  <<  \n",
           (pendpos >= 0 ? pendpos : termpos), curpos);
        printf("          command = %d\n", cmd->funct);
        getchar();
        }
    rflag--;
#endif
    }
```

```
/* NAME
 *    movcur - MOVe CURsor (internal)
 *
 * SYNOPSIS
 *    static void movcur(newpos)
 *       int newpos;
 *
 * DESCRIPTION
 *   Moves the terminal cursor to the specified position and
 *   resets any pending position.
 */


static void movcur(newpos)
   register int newpos;
   {
   register int dr, dc, nr, nc;

   if (newpos != termpos)
      {
      dr = (nr = ROW(newpos)) - ROW(termpos);
      dc = (nc = COL(newpos)) - COL(termpos);
      if (termpos >= 0  &&  dr == 0)
         {
         tputs("\33[", chan);
         if (dc > 0)
            {
            tputnum(dc, chan);
            tputc('C', chan);
            }
         else
            {
            tputnum(-dc, chan);
            tputc('D', chan);
            }
         }
      else if (termpos >= 0  &&  (dc == 0  ||  nc == 0))
         {
         if (dc != 0) tputc('\r', chan);
         tputs("\33[", chan);
         if (dr > 0)
            {
            tputnum(dr, chan);
            tputc('B', chan);
            }
```

```
            else
                {
                tputnum(-dr, chan);
                tputc('A', chan);
                }
            }
        else
            {
            tputs("\33[", chan);
            if (nr > 0) tputnum(nr+1, chan);
            tputc(';', chan);
            if (nc > 0) tputnum(nc+1, chan);
            tputc('H', chan);
            }
        termpos = newpos;
        }
    pendpos = -1;
    }
```

```
/* NAME
 *    setatr - SET ATtRibutes (internal)
 *
 * SYNOPSIS
 *    void setatr(atr)
 *       int atr;
 *
 * DESCRIPTION
 *    Sets the specified terminal attributes.
 */

static void setatr(atr)
   int atr;
   {
   register int i;


   tputs("\33[", chan);
   i = 0;
   while ((i = ffbsa(&atr, GRSIZ, i)) > 0)
      {
      tputc(';', chan);
      tputnum(i, chan);
      }
   tputc('m', chan);
   }

#ifdef PRINTER
/* #include PRINTER */
#include "CI600.C"
#endif
```

END

8-87

DTIC